Lab Overview

In this lab assignment, you will do the following:

- Add a serial EEPROM to your hardware. Implement a bit-banged interface to the EEPROM.
- Add an LCD to your hardware. Implement a memory mapped I/O interface to the LCD and use C pointers to access the LCD as a memory-mapped peripheral.
- Write device drivers for the EEPROM and LCD.
- Write assembly and C programs to implement a user interface and perform user tasks.
- Gain experience in code integration and how to use embedded C including interrupts.
- Continue learning about the ARM architecture.

Students must work individually and develop their own original and unique hardware/software.

The Part 1 Elements of this lab assignment should be done (milestone only) by **Friday, Nov. 3, 2023**.
(Part 1 is not late if done by the Part 2 Elements due date)
The Part 2 Elements of this lab assignment are due by **Friday, Nov. 10, 2023**.
The Part 3 Elements of this lab assignment are due by **Friday, Nov. 17, 2023**.
The final submission due date (when all files must be turned in) is **11:59pm Sunday, Nov. 19, 2023**.

The cutoff date for this lab is **Saturday, Dec. 2, 2023**.

Software files must be submitted by 4pm on the signoff day to be considered on time.

This lab is weighted as ~20% of your course grade.

Note: The LCD interface should be implemented with the 8051 hardware due to the 5V logic on the LCD. The I^2C EEPROM interface may be implemented with either the 8051 or the ARM hardware.

Required elements are necessary in order to meet the minimum requirements for the lab. Supplemental/challenge elements of the lab assignment may be completed by the student to qualify for a higher grade, but they do not have to be completed to successfully meet the requirements for the lab.

ECEN 5613 students will have to complete some supplemental elements and show differentiation in the quality of their work in order to compete for the highest grades. To earn maximum credit and avoid any late penalties, ECEN 5613 students must obtain a TA's signature on their work by the specified signature due dates for required and supplemental elements.

All items on the signoff sheet must be completed to get a signature, but partial credit is given for incomplete labs. Receiving a signature on the signoff sheet does not mean that your work is eligible for any particular grade; it merely indicates that you have completed the work at an acceptable level.

NOTE: **The quality of your user interfaces and demo will impact your score on the lab.** Your goal should be to ensure that the user has a successful and positive experience with your software. Your code should handle error conditions gracefully (e.g. user input values outside the allowed range). Top scores are reserved for those students who submit outstanding implementations, including coding style.

Lab Details

1. Review the homework assignments associated with this lab.
2. Reserve space on your board for both the EEPROM wire wrap socket and the large LCD module.
3. **[Part 1 Required Element[1]]** Read the EEPROM Guide "Adding an NM24C04 (or NM24C16) EEPROM to your board", available on the course web site. It has ideas and information on interfacing to the I^2C EEPROM.

   Read the data sheet for the serial EEPROM included in your parts kit (e.g. Microchip 24LC16 or Fairchild-National Semiconductor NM24C16). You may also read Fairchild App Note AN-794.

   [Optional, but recommended] Review Microchip app notes AN536, AN572, AN614 and AN709.

   Design and implement your bit-banged EEPROM circuit. Your EEPROM should be connected to two unused port pins. Note that since you are connecting to the EEPROM using port pins, the EEPROM does not consume any processor address space.

   NOTE: When asking the TA's and instructor for help with your code, be sure to attach all required/relevant files to your e-mail. Include commented source files (.c, .h), and the listing and memory usage files (for SDCC, the .rst, .mem, and .map files). If your question involves external peripherals (even NVSRAM), you also need to attach a PDF or sketch of your schematic.

4. **[Part 1 Required Element[1]]** Implement an EEPROM I^2C device driver with the ability to bit-bang write and read a byte at any EEPROM I^2C address using function calls from C. The underlying drivers may be in assembly if you wish, but the functions must be accessible from C. It does not matter what you name the functions. For example, you might implement the following two functions.

```
int eebytew(addr, databyte)  // write byte, returns status
int eebyter(addr)            // read byte, returns data or status
```

   NOTE: A variety of I^2C routines and libraries written in C are available on the web – a quick web search would be beneficial. You may use these libraries as long as your code contains clear documentation of how you obtained, utilized and/or modified them. Each of your code files must have a file header which identifies all authors of the code. (You already know this is the standing expectation in this class with regard to borrowed code.) **You must have a complete understanding of how all the code works**.

   Verify that you can write data to and read data from the EEPROM using your I^2C device driver and **verify the stored data is correct after cycling power**.
   Your code should not ignore ACK's during I2C transactions.

5. **[Part 1 Required Element[1]]** Use a logic analyzer to prove that your byte write function sends the correct signals and has the correct I^2C timing. Note that the LogicPort logic analyzer has an I^2C interpreter that you might find useful as you debug your I^2C bus transactions.

   [Optional] Use an oscilloscope to trigger on a write or read frame. Display SCL and SDA on the oscilloscope screen and verify that the transaction is for the address you intended. Verify that your rise and fall times fall within the limits given in the I^2C specification and/or EEPROM data sheet.

   - **A simple hand sketch or a logic analyzer screen capture of these timing relationships and values must be shown to and discussed with the TA during signoff and turned in with your lab, along with your timing analysis.**

6. **[Part 1 Required Element[1]]**

NOTE: For this lab, as much of your code as possible should be integrated – this will provide experience with integrating much functionality into a single program, and will also reduce signoff times since only a single program must be stored in the flash memory on your processor. Your demo and submission should be one well-integrated program, but the program can be modularized and consist of multiple code and header files.

Provide a well-designed menu on the PC terminal emulator screen which allows the user to:

- **Write Byte**: Enter an EEPROM address and a byte data value in hex. If the address and data are valid, store the data into the EEPROM. The program must allow any hex value from 0x00 to 0xFF to be programmed into **any** location in the EEPROM. Do not make the user type in "0x" before the address or data hex value.
- **Read Byte**: Enter an EEPROM address in hex. If the EEPROM address is valid, display on the PC screen in hex the contents of the EEPROM address, using the format "AAA: DD". Do not make the user type in "0x" before the address hex value.
- **Hex Dump**: Enter a start address and end address in hex. If the entered values are valid, read the contents of the EEPROM from the start address to the end address and display the data on the PC screen in hexadecimal, with a maximum of 16 bytes of data per line, in the following format:
  AAA: DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD
  This format is similar to what you see when using the device programmer or when dumping memory contents using PAULMON2, where AAA is the starting address (in hex) for each block of 16 data values DD (in hex). The first memory cell in the EEPROM is address 0x000. You should be able to leverage code from Lab #3. Do not make the user type in "0x" before the address hex values.
- **Reset EEPROM**: Call a function named `eereset()` in your EEPROM I²C device driver:

```
// Name: eereset()
// Description: Performs a software reset of the I2C EEPROM using an
// algorithm that conforms to Microchip application note AN709.
void eereset()
```

  Use a logic analyzer to prove that eereset sends the correct sequence and has the correct I²C timing. Show the trace on the logic analyzer to the TA during signoff. You do **not** need to print/submit the trace with your report.

  Demonstrate that you understand application note AN709 and the functionality of the eereset code during your sign-off.

The user must be able to execute menu items in any order (the program should not include any dependencies on the order in which a user selects menu items).

Note: Students must utilize version control and save code to a remote server after milestones are met during embedded software development. Keep code private.

COMPLETE AND SUBMIT ALL THE ITEMS ABOVE FOR THE PART 1 ELEMENTS INFORMAL CHECKPOINT.

---

[1] Required elements are necessary in order to meet the requirements for the lab. Supplemental, optional, and challenge elements of the lab assignment may be completed by the student to qualify for a higher grade, but they do not have to be completed to successfully meet the minimum requirements for the lab.

7. **[Part 2 Required Element[1]]** Review the data sheets for the Optrex DMC 20434 LCD and the SED1278F (or Hitachi HD44780U) LCD controller. Review the document "SED1278F Technical Manual Errata".

   Refer to the LCD Guide ("Adding an LCD (with an HD44780 LCD controller) to your board") available on the course web site for further ideas and information on interfacing to the LCD module. It contains some very important notes, including one regarding errors in the LCD data sheet.

   Devise a way to securely mount the LCD **and** properly connect all of the data lines to your board. It may take you a little time to devise a good physical interface, so don't wait too long before getting started. Wire can be used to easily attach the LCD to your board without requiring any drilling (remember the previous warnings against drilling holes in the PCB).

   Data Lines: Most LCDs will have only 14 pins. LCDs with a backlight will have 16 pins, with two for the backlight. One option for connecting data lines is to use a 14-pin strip header or SIP wire wrap socket. You may also attach the LCD through a ribbon cable to a 14- or 16-pin DIP socket on your board. A sturdy data line connection using a strip header can make it easy to mount the LCD.

   Design and implement your LCD circuit. Your LCD must be memory mapped in the address space reserved for peripherals (this is an example of memory mapped I/O), and will be accessible using the MOVX instruction (and via a pointer variable in C). The LCD contrast (V_{EE}) can sometimes be grounded, but you probably need to use a potentiometer or resistor divider to control the contrast so that you can see characters on the screen. The LCD in the parts kit has 14 pins/lines which must be connected; any additional pins on the module are No Connect.

   The eight data signals on the LCD must be connected to the data lines on the processor (e.g. Port 0 of the 8051); you may not use GPIO pins for these data signals.
   Ensure that the E signal on the LCD is high **only** when reading from or writing to the LCD.

8. **[Part 2 Required Element[1]]** Implement an LCD device driver with the following C functions:

- ```
  // Name: lcdinit()
  // Description: Initializes the LCD (see Figure 25 on page 212
  //  of the HD44780U data sheet).
  void lcdinit()
  ```

- ```
  // Name: lcdbusywait()
  // Description: Polls the LCD busy flag. Function does not return
  //  until the LCD controller is ready to accept another command.
  void lcdbusywait()
  ```

- ```
  // Name: lcdgotoaddr()
  // Description: Sets the cursor to the specified LCD DDRAM address.
  //  Should call lcdbusywait().
  void lcdgotoaddr(unsigned char addr)
  ```

- ```
  // Name: lcdgotoxy()
  // Description: Sets the cursor to the LCD DDRAM address corresponding
  //  to the specified row and column. Location (0,0) is the top left
  //  corner of the LCD screen. Must call lcdgotoaddr().
  void lcdgotoxy(unsigned char row, unsigned char column)
  ```

- ```
  // Name: lcdputch()
  // Description: Writes the specified character to the current
  //  LCD cursor position. Should call lcdbusywait().
  void lcdputch(char cc)
  ```

- ```
  // Name: lcdputstr()
  // Description: Writes the specified null-terminated string to the LCD
  //  starting at the current LCD cursor position. Automatically wraps
  //  long strings to the next LCD line after the right edge of the
  //  display screen has been reached. Must call lcdputch().
  void lcdputstr(char *ss)
  ```

- ```
  // Name: lcdclear()
  // Description: Clears the LCD using the HD44780 Clear display
  instruction.
  void lcdclear()
  ```

NOTE: I prefer that you write your own code for these routines. However, a variety of LCD routines and libraries suitable for SDCC are available on the web. You may use these libraries as long as your code contains clear documentation of how you obtained, utilized and/or modified them. Each of your code files must have a file header which identifies all authors of the code. **You must have a complete understanding of how all the code works**.

Write a simple program that uses your LCD driver to prove that the required functions are implemented correctly. Choose the sequence carefully so that it is easy for the TA to see that each function did its job correctly during the demonstration. This program is just test code and does not need to be completely robust, as long as it adequately demonstrates each of the LCD functions above.

9. **[Part 2 Required Element[1]]** Using a logic analyzer, prove that your LCD control signal timing is correct. Show the timing between the E, RS, RW, and data signals as measured at the LCD interface.

- **A logic analyzer screen capture or a simple hand sketch of these timing relationships and values must be shown to and discussed with the TA during signoff and turned in with your lab, along with your timing analysis.**

You should also be able to prove that the LCD E control signal goes high only when the LCD is being accessed. You can verify this by running code which does not access the LCD and by triggering the logic analyzer on E going high. If E goes high during this test, then your implementation is incorrect. You may also be able to test this by using Paulmon2.
  ```

10. **[Part 2 Required Element[1]]:**

NOTE: The code for this element must be integrated with the previous LCD code above. Your demo and submission should be one well-integrated program.

- In the bottom right of the LCD, continuously display the elapsed time since your program started running using the format "MM:SS.S", where MM is the number of elapsed minutes and SS.S represents the seconds to one-tenth of a second accuracy. For example, 5.1 seconds would be displayed as "00:05.1" and 64.3 seconds would be displayed as “01:04.3”.
- Provide **Clock** menu options to stop the elapsed time clock, to restart the clock, and to reset the clock back to "00:00.0".

NOTE: Make sure that the cursor location is correctly stored before and restored after any ISRs.

NOTE: If using SDCC, read the "interrupt" sections of the SDCC user manual carefully, and remember the correct use of 'volatile' and ‘critical’. Be careful when using variables from within the context of an ISR. This includes any functions that your ISR calls. Do not use printf/sprintf in an ISR (note that printf and sprintf share code).

NOTE: This element is an addition to the previous required LCD element**. The two code elements must be integrated together.** The elapsed timer must work correctly while simultaneously allowing all the menu options in the previous C program to work correctly.

COMPLETE ALL THE ITEMS ABOVE FOR THE PART 2 ELEMENTS SIGNOFF SUBMISSION.

COMPLETE ALL THE ITEMS BELOW FOR THE PART 3 ELEMENTS SIGNOFF SUBMISSION.

11. **[Part 3 Supplemental Element[1]]**

Provide a well-designed menu on the PC terminal emulator screen with the following options:

- **LCD RAM Dump**: Reads the entire contents of the LCD DDRAM and displays it in hex on the PC screen in a clean and logical format. Then reads the entire contents of the LCD CGRAM and displays it in hex on the PC screen in a clean and logical format.
- **Create Custom Character**: Design and implement C routines which allow the creation of custom LCD characters using CGRAM. Implement the following function:

```
// Name: lcdcreatechar()
// Description: Function to create a 5x8 pixel custom character with
// character code ccode (0<=ccode<=7) using the row values given in
// the 8-byte array row_vals[].
void lcdcreatechar(unsigned char ccode, unsigned char row_vals[])
```

Provide a way for users to enter and display their own customer characters. A good custom character generation routine user interface should:

(a) accept values from the user representing the pixel pattern for each row of the custom character (the design can choose to allow either strings of bits or hex values representing each row of the character)

(b) display the current state to the user on the terminal screen after each row is entered

As part of your demo, show that you have created some fun logo using custom characters. For example, you could use several custom characters grouped together to create a pixel map of the CU logo.

12. **[Part 3 Supplemental Element[1]]: (8051 or ARM)**

    Explore the SPI protocol and the use of a SPI controller.

    Obtain a discrete ADC or DAC device that has a SPI interface and connect it to the 8051 or ARM board. Configure the processor to communicate with the device using the processor's integrated SPI controller and demonstrate that SPI device's capabilities. Be careful not to exceed the max voltage level specification for the input pins.

    Use a logic analyzer to study a SPI transaction and understand the protocol. Show the trace on the logic analyzer to the TA during signoff. You do **not** need to print/submit the trace with your report.

    During your signoff, be able to compare SPI and I^2C and discuss each protocol's relative advantages.

13. **[Part 3 Supplemental Element[1]]:**

    Continue learning about the ARM architecture.

    a) Demonstrate that you can successfully build and execute code that demonstrates at least two additional ARM processor/dev board features that you did not previously demonstrate earlier in the semester. Include an interrupt service routine in your ARM code for Lab #4.

       STM32 BOARD: Some implementation ideas include but are not limited to: Interfacing the onboard LSM303DLHC linear acceleration sensor via the I2C1 peripheral, interfacing the onboard L3GD20 gyroscope via the SPI1 peripheral, configuring the onboard RTC, Watch Dog Timers (Independent WDG and Window WDG), exploring and implementing basic DMA transfer, integrating low power modes with some existing functionality (measuring current consumption by connecting a multimeter across JP2), implementing CRC check using the onboard peripheral.

    b) Explain your key learnings to the TA's.

14. **[Part 3 Supplemental Element[1]]: (8051 or ARM)**

    NOTE: If this element is implemented on the same system as your previous I^2C code, the code for this element should be integrated into the previous I^2C programs above and your demo and submission should be one well-integrated program.

    Read the PCF8574 I^2C I/O expander data sheets and application notes available from the course web site. Integrate the chip into your embedded system, sharing the I^2C bus with the EEPROM, and prove that you can configure some of its I/O pins to work as inputs and other pins to work simultaneously as outputs (at least one input and one output). Your parts kit already included a 16-pin wire wrap socket that could be used with the I2C expander chip. You can purchase another wire wrap socket if necessary.

    Choose some easy mechanism to demonstrate your I/O expander. For instance, read the state of one PCF8574 pin that is configured as an input, then invert that value and write it back out to the output pin on the PCF8574. Use the interrupt signal from the I/O expander to notify the processor when the input changes. Make sure you choose appropriate triggering (level- or edge-) for the interrupt on the processor. Provide a basic software driver that allows you to configure the pins individually as inputs or outputs, and also to check the status of the pins and to write to the pins that are outputs. Remember to use a bit mask in software when interacting with specific pins on the chip.

    Note: Your choices of input and output pins can be hard coded for this element; you don't need to create a user interface that allows the user to arbitrarily select which pins are inputs and outputs.

Submission Questions

15. **[Part 3 Required Element[1]]** As part of your submission, provide answers to the following:

a) What operating system (including revision) did you use for your code development?

b) What compiler(s) (including revision) did you use?

c) What exactly (include name/revision if appropriate) did you use to build your code (an IDE, make/makefile, or command line)?

d) Did you install and use any other software tools to complete your lab assignment?

e) Did you experience any problems with any of the software tools? If so, describe the problems.

NOTE: Make copies of your code, SPLD code, and schematic files and save them as an archive. You will need to submit the Lab #4 files electronically at the end of the semester.

Submission Instructions

Instructions: Print your name and sign the honor code pledge. Separate the signoff sheet from the rest of the lab and turn in a scan of the signed form, the items in the checklist below, and the answers to any applicable lab questions in order to receive credit for your work. No cover sheet please. **Submit all items electronically via Canvas (https://canvas.colorado.edu).**

Please follow the instructions given below:

1. Create a folder called "lastname_lab_4" where "lastname" is your last name. You will create subfolders inside this unzipped folder, ending with a folder structure like the following:
    - lastname_lab_4
        - schematic
        - spld_code
        - 8051_code
        - stm32_code
        - lab_writeup
2. Please include a legible and easy-to-view copy of your complete and accurate schematic (all old/new components shown) in pdf format inside the sub-folder named "schematic".
3. Include your spld source file (.pld file) and any spld simulation file (.si file) in the sub-folder "spld_code".
4. Submit all of your 8051 code files in the sub-folder named "8051_code". Include full copy of fully, neatly, clearly commented source code (including C and header files, and .RST, .MEM, and .MAP files). Ensure your code is neatly formatted and easy to read, and that each and every source file has header comments that identify the author and any leveraged code the file contains - **there will be deductions if this information is not provided for all files.** Code must be well organized in folders. If you submit a PDF of your code to improve its appearance, be aware that you must also submit the original source code files as well. Your lab grade will not be released unless you have submitted your original source files.
5. Submit all of your ARM code files in the sub-folder named "stm32_code". Ensure that these files are neatly formatted and easy to read.
6. Submit your write-up in the sub-folder named "lab_writeup" either in Word or PDF format. Your write-up should follow these guidelines:
    - Include scan of signed and dated signoff sheet as the first page. Include the backside of the signoff sheet as well if there are comments written by the TAs. Do not add any cover sheet.
    - Write-up should be concise and less than 5 pages (excluding the signoff sheet). Use only font size 11 or 12 to maintain legibility. Do not describe the objectives of the lab or any redundant information. Submit the SDCC command line options you used when building your code. If you used a makefile, submit a copy of that makefile.
    - Include any important analyses and timing diagrams (e.g. scan of timing diagrams and analyses for the EEPROM and LCD interfaces)
    - Include all the calculations that are required to be done for answering the questions that have been asked in the sign-off sheets or lab assignment.
    - Include screen-shots (if any) in PDF format in this folder itself.
    - Comment on any significant learnings from the lab assignment.
    - Include the answers to any submission questions (e.g. details on software environment and tools).
    - Must include clear high-resolution pictures of top and bottom sides of the 8051 board you assembled for this lab, including wiring and labels; wire wrapping and soldering must be clearly visible when you zoom in – **make sure your pictures are in focus**. Use JPEG format.
7. Submit a zipped version of the folder as "lastname_lab_4.zip" as an attachment via the Canvas interface. Please use only the ".zip" file format when submitting files.
8. Please contact the TA's or instructor in case you have any doubts regarding submissions.

NOTE: Make and save archive copies of your code, SPLD code, and schematic files. You need to submit the Lab #4 files electronically, now and at the end of the semester.

ECEN 5613 **Lab #4 Signoff Sheet** **Fall 2023**

You will need to obtain the signature of your TA on the following items in order to receive credit for your lab assignment. Print your name below, sign the honor code pledge, and then demonstrate your working hardware & firmware in order to obtain the necessary signatures.

Student Name: _____

Honor Code Pledge: "On my honor, as a University of Colorado student, I have neither given nor received unauthorized assistance on this work. I have clearly acknowledged work that is not my own."

Student Signature: _____

Signoff Checklist

Part 1 Elements

- ❑ Pins and signals labeled and decoupling capacitors present on board
- ❑ C code for EEPROM functional, contents present after power cycle
- ❑ I^2C diagram/timing analysis

_____

TA signature and date

Part 2 Elements

- ❑ LCD functional, C code for basic LCD routines functional
- ❑ LCD control signal timing meets specifications (logic analyzer trace/diagram, analysis)
- ❑ Elapsed time stop, restart, reset to "00:00.0":
- ❑ Good integration with previous code, all functions work, no irregularities _____

Part 3 Required and Supplemental Elements

- ❑ LCD Hex/DDRAM/CGRAM dumps, custom LCD characters, fun logo
- ❑ SPI interface, logic analyzer trace, compare with I^2C.
- ❑ ARM code development, 2 new features, ISR
- ❑ PCF8574 I^2C I/O Expander, input, output, ISR _____

FOR TA/INSTRUCTOR USE ONLY Part 1 Elements	Not Applicable	Poor/Not Complete	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	☐	☐
Hardware physical implementation	☐	☐	☐	☐	☐
Required Elements functionality	☐	☐	☐	☐	☐
Sign-off done without excessive retries		☐	☐		
Student understanding and skills	☐	☐	☐	☐	☐
Overall Demo Quality (Part 1 elements)	☐	☐	☐	☐	☐

FOR TA/INSTRUCTOR USE ONLY Part 2 Elements	Not Applicable	Poor/Not Complete	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	☐	☐
Hardware physical implementation	☐	☐	☐	☐	☐
Required Elements functionality	☐	☐	☐	☐	☐
Sign-off done without excessive retries		☐	☐		
Student understanding and skills	☐	☐	☐	☐	☐
Overall Demo Quality (Part 2 elements)	☐	☐	☐	☐	☐

FOR TA/INSTRUCTOR USE ONLY Part 3 Elements	Not Applicable	Below Expectation	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	☐	☐
Hardware physical implementation	☐	☐	☐	☐	☐
Required Elements functionality	☐	☐	☐	☐	☐
Supplemental Elements functionality	☐	☐	☐	☐	☐
Sign-off done without excessive retries		☐	☐		
Student understanding and skills	☐	☐	☐	☐	☐
Overall Demo Quality (Part 3 elements)	☐	☐	☐	☐	☐

TA/Instructor Comments □ □ □